# 数字电路与数字系统实验

## EX07:存储器

191220029 傅小龙

周一5-6节班

[1830970417@qq.com](mailto:1830970417@qq.com)

2020年10月14日

# **目录**

1. [实验内容](#_实验内容)……………………………………………………………………………………3
   1. [实验要求](#_1.1实验要求)………………………………………………………………………………3
   2. [实验工具](#_1.2实验工具)………………………………………………………………………………3
2. [实验过程](#_实验过程)……………………………………………………………………………………4

2.1 RAM1

2.1.1 [模型概述](#_2.1模型概述)………………………………………………………………………4

2.1.2 [数字抽象](#_2.2数字抽象)………………………………………………………………………4

2.1.3 [建立模型](#_2.3建立模型)………………………………………………………………………4

2.2 RAM2

2.2.1 [模型概述](#_2.2.1模型概述)………………………………………………………………………5

2.2.2 [数字抽象](#_2.2.2数字抽象)………………………………………………………………………5

2.2.3 [建立模型](#_2.2.3建立模型)………………………………………………………………………6

2.3 [分析/综合](#_2.4分析/综合)……………………………………………………………………………10

2.4 [分配引脚](#_2.6分配引脚)………………………………………………………………………………12

2.5 [全编译](#_2.7全编译)…………………………………………………………………………………13

三、[实验总结](#_实验总结)……………………………………………………………………………………13

四、附……………………………………………………………………………………………14

4.1关于思考题……………………………………………………………………………14

4.2关于存储器实例分析…………………………………………………………………14

4.3 1Hz分频器的Verilog实现…………………………………………………………15

4.4七段数码管编码器的Verilog实现…………………………………………………16

#### 实验内容

##### 1.1实验要求

在一个工程中完成如下两个存储器。两个存储器的大小均为 16×8，即每个存储器共有 16 个存储单元，每个存储单元都是 8 位的，均可以进行读写。

RAM1：采用下面的方式进行初始化，输出端有输出缓存，输出地址有效后，等时钟信号的上升沿到来时才输出数据。

1 initial

2 begin

3 $readmemh("D:/digital\_logic/mem1.txt", ram, 0, 15);

4 end

初始化数值详见2.1.3节.

RAM2：利用 IP 核设计一个双口存储器，利用.mif 文件进行初始化，十六个单元的初始化值分别为：0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6,0xf7,0xf8,0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff。

此两个物理上完全不同的存储器共用时钟、读写地址和写使能信号，当写使能有效时，在时钟信号的有效沿写入数据；当写使能信号无效时，在时钟信号的有效沿输出数据。适当选择时钟信号和写使能信号，以能够分别对此两个存储器进行读写。

##### 1.2实验工具

软件环境：

设计、编译、仿真：Quartus Prime Version 17.1.0 Build 590 10/25/2017 SJ Lite Edition

DE10\_Standard\_SystemBuilder

硬件环境： DE-10 Standard开发平台

FPGA芯片： Cyclone V 5CSXFC6D6F31C6

#### 实验过程

##### 2.1 RAM1

###### 2.1.1模型概述

该存储器拥有16\*8大小的内存，可进行读写操作.当写使能信号有效时，在时钟信号的上升沿写入数据；当写使能信号无效时，在时钟信号的上升沿输出数据.输出端有输出缓存. 时钟信号采用1Hz分频器的输出信号.

###### 2.1.2数字抽象

1. 输入:

时钟信号clk： 与时钟信号连接.

写使能信号we: 控制存储器进行读/写操作.

读地址[3:0]inaddr: 进行读操作时要输出的数据的地址.

写地址[3:0]outaddr: 进行写操作时要写入的数据的地址.

数据输入[7:0]din: 8位二进制码的输入.

1. 输出:

数据输出[7:0]dout: 8位二进制码的输出

下表\图给出了以上输入输出信号在DE10平台对应的信号：

|  |  |  |
| --- | --- | --- |
|  | 信号名称 | DE-10平台信号 |
| 输入 | clk | LEDR[9]① |
| we | KEY[0] |
| [3:0]inaddr | SW[9:6] |
| [3:0]outaddr | SW[5:2] |
| [7:0]din | SW[1:0]② |
| 输出 | [7:0]dout | [7:0]dout0③ |

①:LEDR[9]为分频器产生的1Hz时钟信号输出

②:由于FPGA开发板的输入端数量有限，写入数据宽度约束为2位数据.

③:dout0在顶层文件中为wire[7:0]型变量，用以接收存储器的数据输出，并连接至七段数码管以在开发板上显示结果.

表 2-1-1:RAM1输入输出信号与DE10平台信号对应关系

###### 2.1.3建立模型

**实现思路：**参照exp07.pdf中的表7-2中的部分Verilog代码和exp07.pdf中的7.2.2节给出的使用外部文件给存储器初始化的介绍，RAM1的实现如下：

module RAM1(clk, we, inaddr, outaddr, din, dout);

input clk;

input we;

input [3:0]inaddr, outaddr;

input [7:0]din;

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6  output reg [7:0]dout;    reg [7:0] ram [15:0];  reg [3:0] out;    initial begin  $readmemh(".\\init\\mem1.txt", ram, 0, 15);  out = 0;  end    always @(posedge clk) begin  if(we)  ram[inaddr] <= din;  out <= ram[outaddr];  dout <= out; //输出缓存  end  endmodule |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |

##### 2.2 RAM2

###### 2.2.1模型概述

通过IP核设计一个双口存储器. 用.mif文件进行初始化.

###### 2.2.2数字抽象

由IP核得到的双口存储器的数据输入输出如下所示：

|  |  |
| --- | --- |
| 1 | RAM2 r2( |
| 2 | .clock(LEDR[9]), |
| 3 | .data(SW[1:0]), |
| 4 | .rdaddress(SW[5:2]), |
| 5 | .wraddress(SW[9:6]), |
| 6 | .wren(KEY[0]), |
| 7 | .q(dout1) //① |
| 8 | ); |

①:dout1在顶层文件中为wire[7:0]型变量，用以接收存储器的数据输出，并连接至七段数管解码器以在开发板上显示结果.

①:LEDR[9]为分频器产生的1Hz时钟信号输出

###### 2.2.3建立模型

参照exp07-7.3.1节内容生成所需的RAM，RAM2的实现步骤如下：

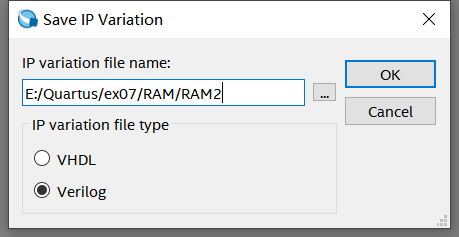


图2-2-1：选择目标文件名

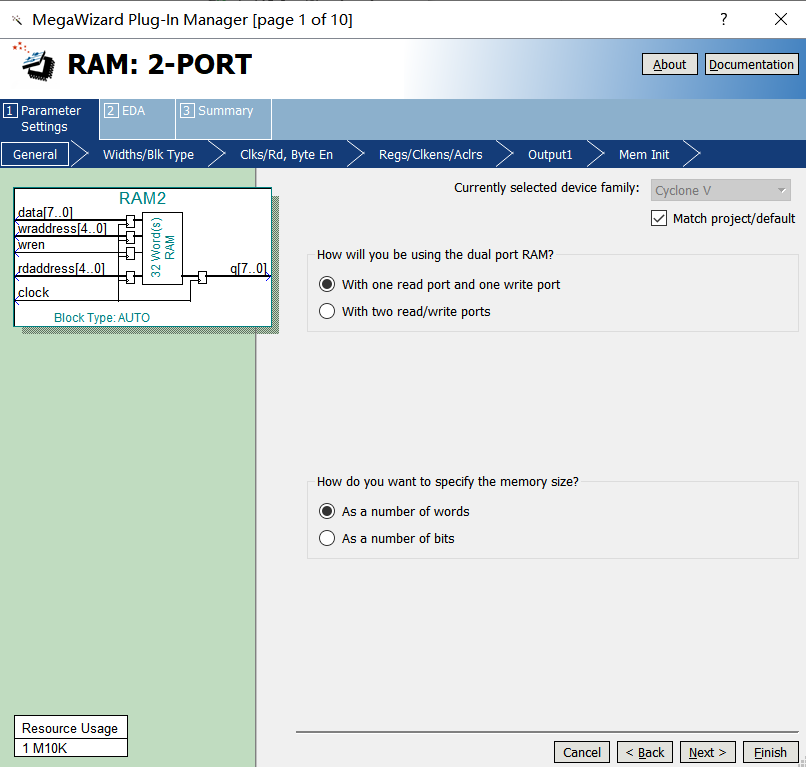


图2-2-2：选择RAM格式

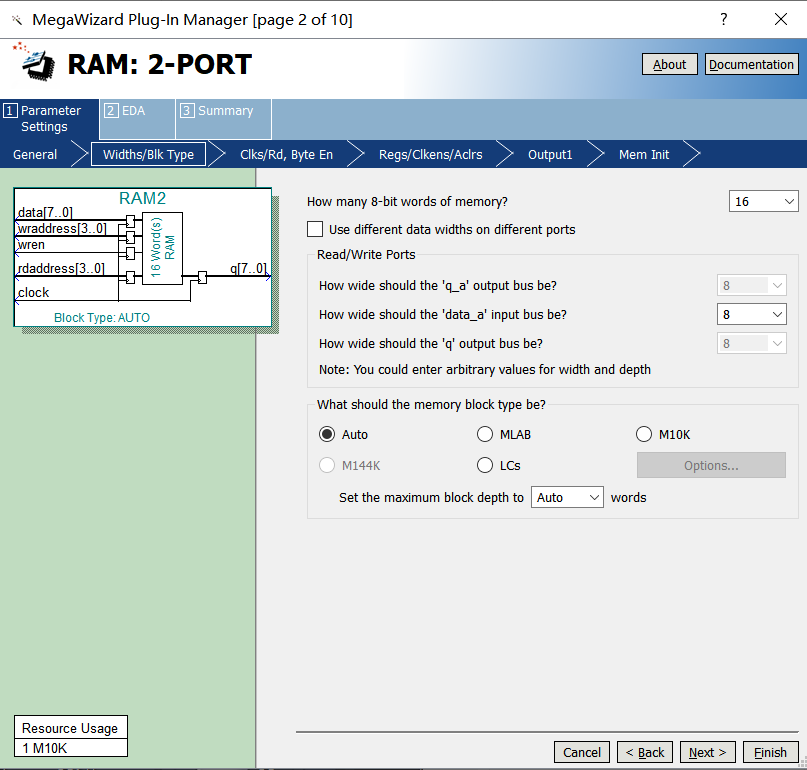


图2-2-3：选择RAM规模

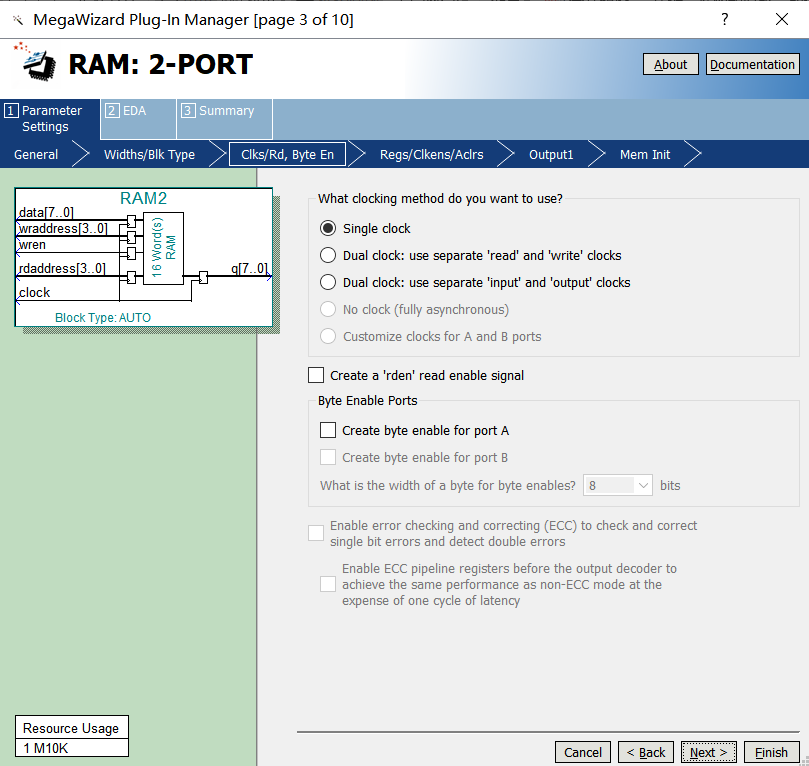


图2-2-4：时钟信号配置

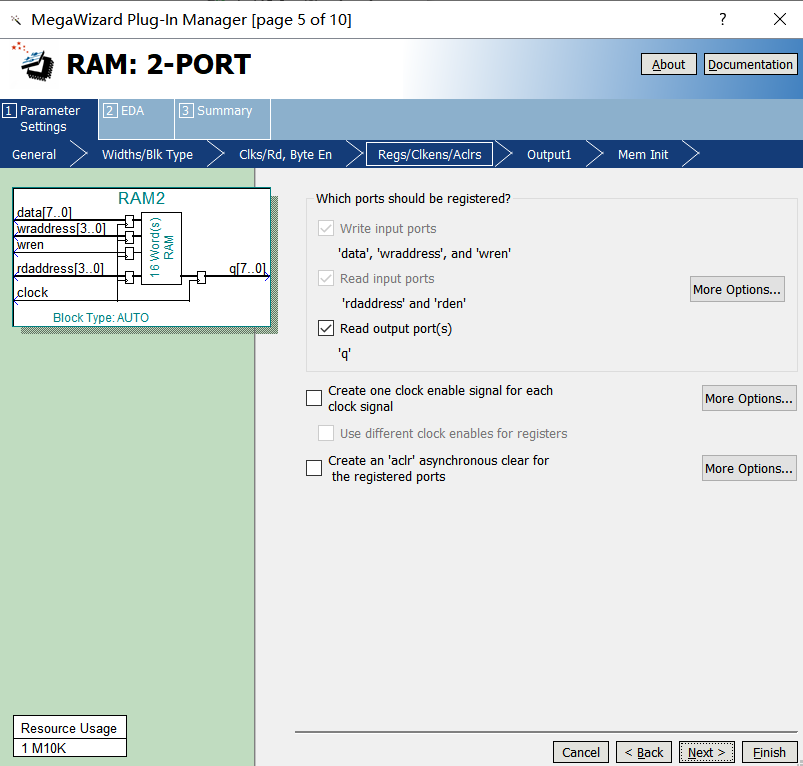


图2-2-5：输出缓存选择

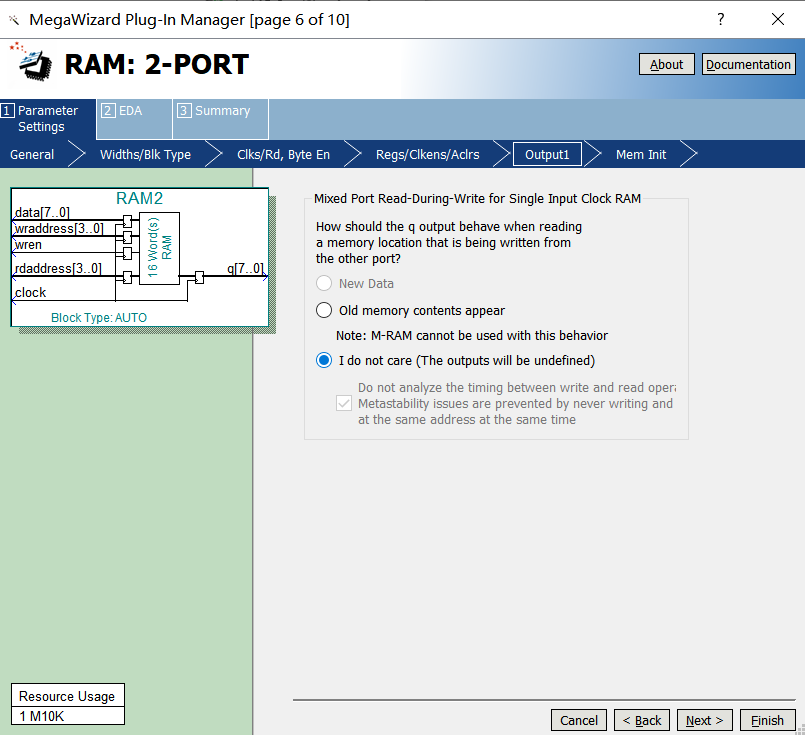


图2-2-6：读写冲突解决

采用.mif文件来对该存储器进行初始化.

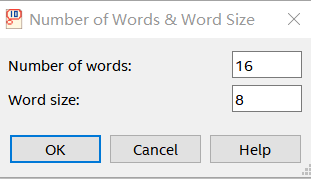


图2-2-6：初始化文件大小选择

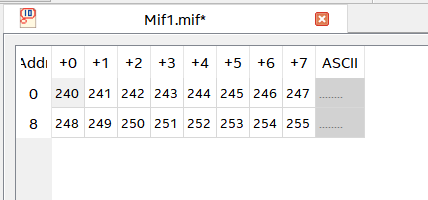


图2-2-7：编辑初始化文件

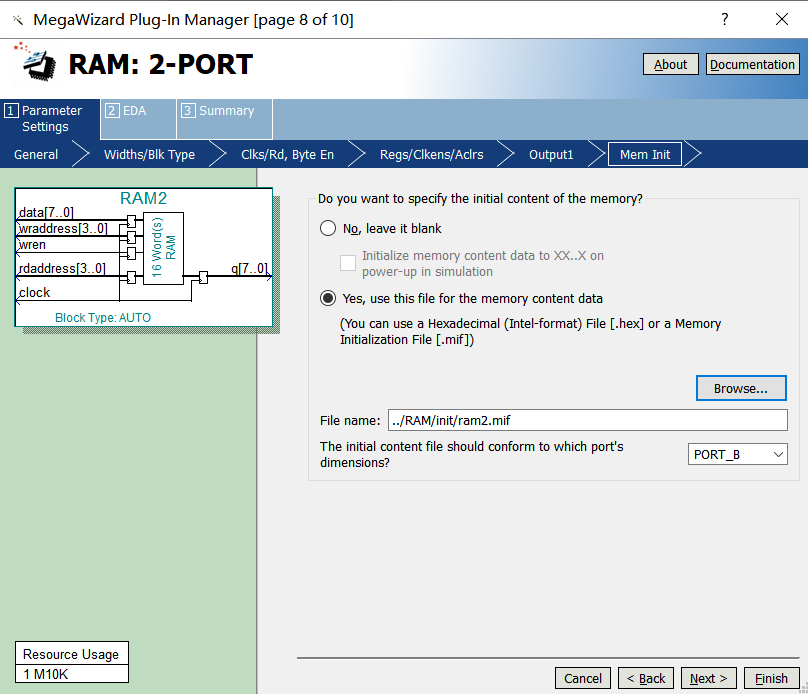


图2-2-8：初始化文件选择

##### 2.3分析/综合

分析/综合实验成功，如下图所示：

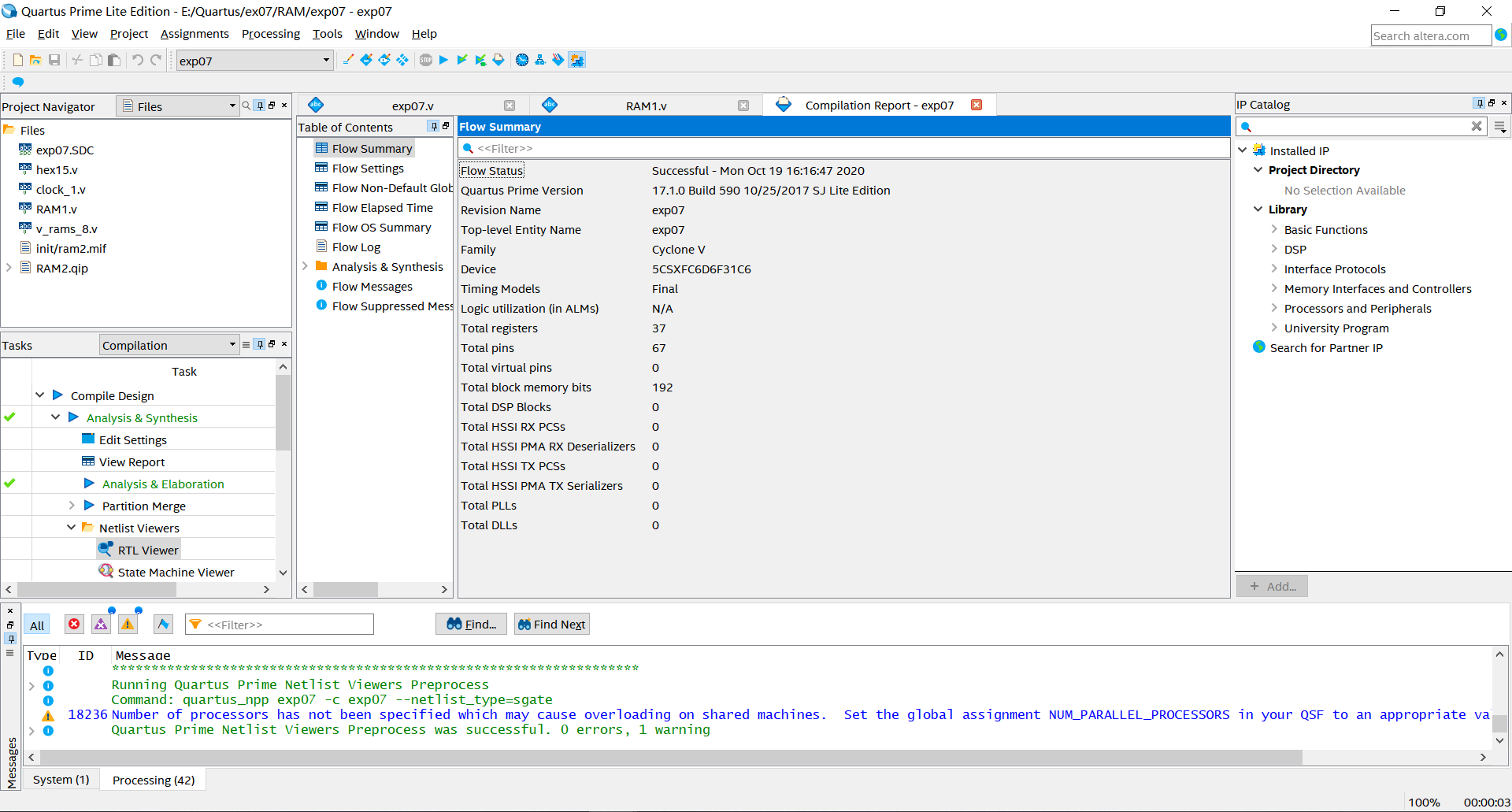
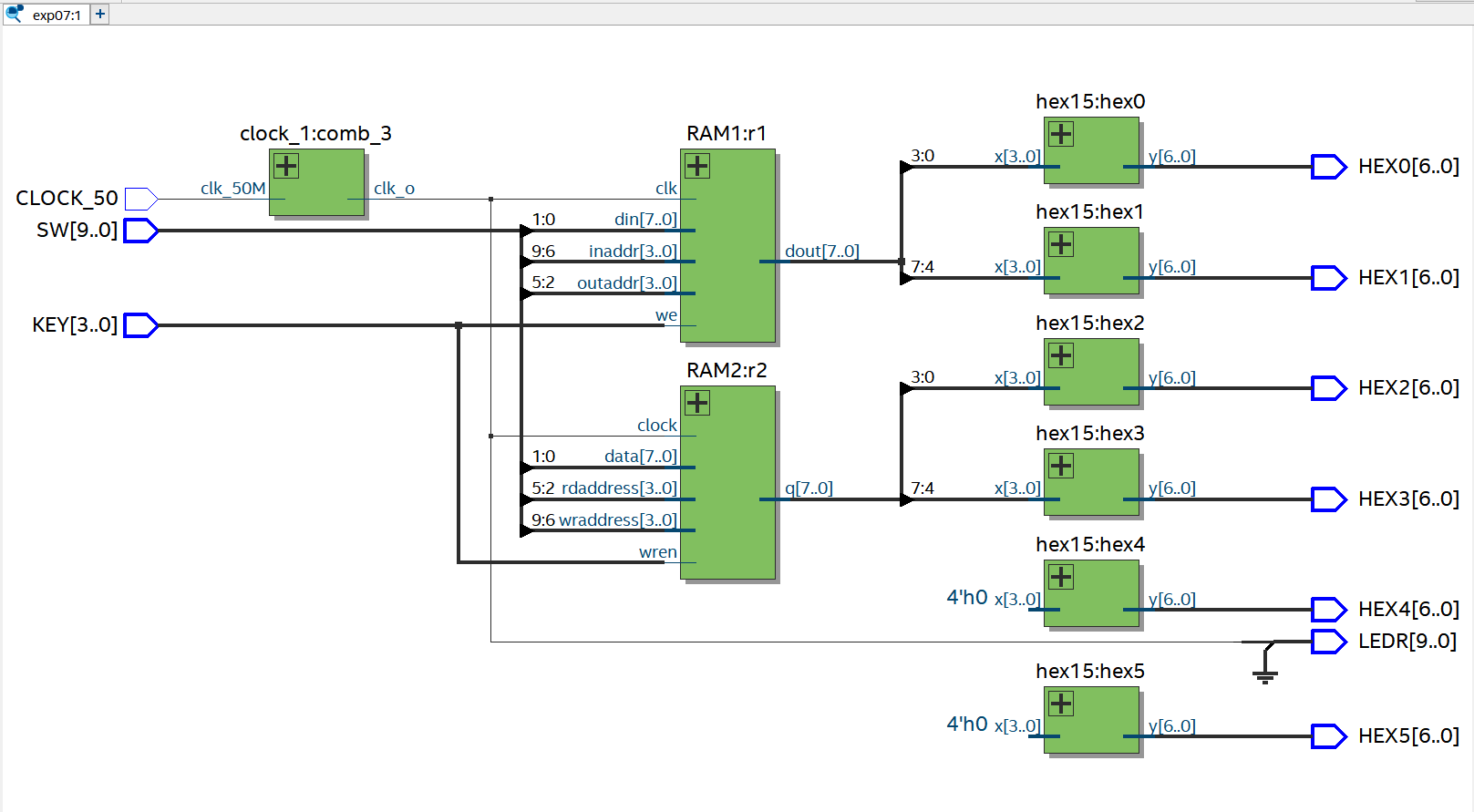
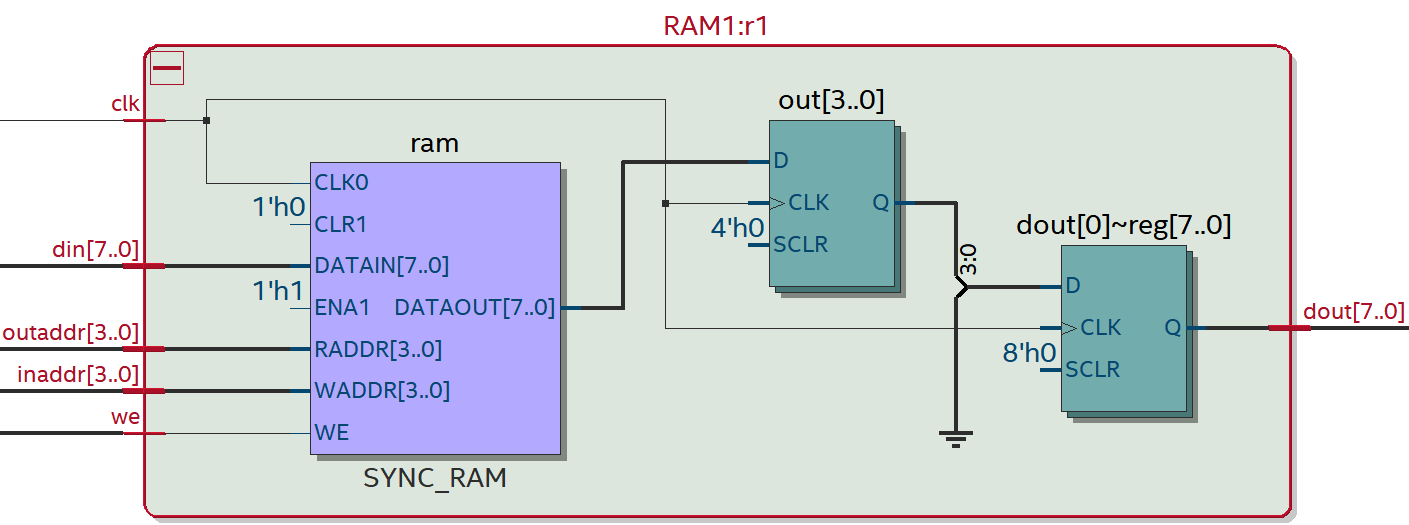


图2-3-1：分析/综合成功





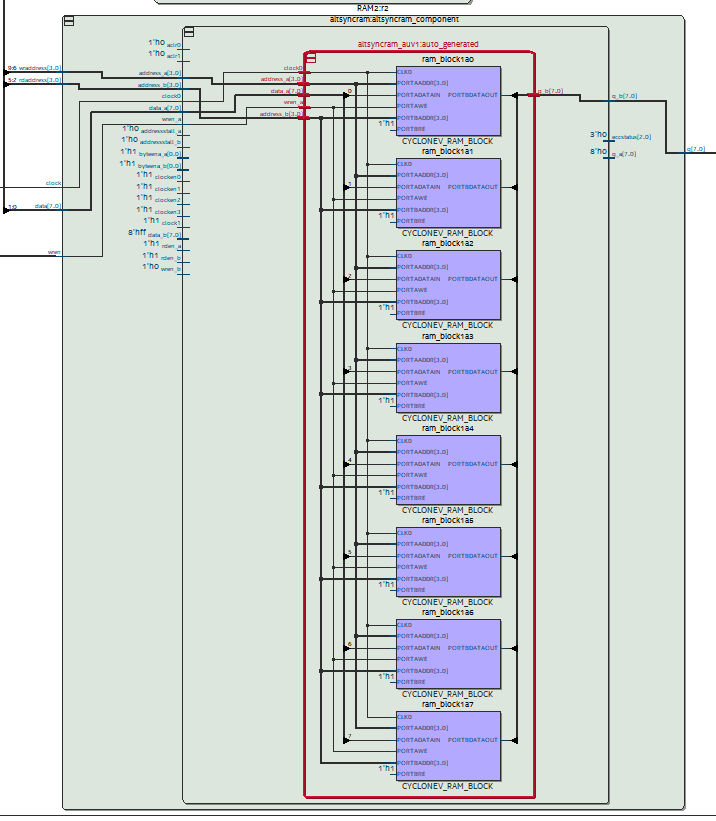
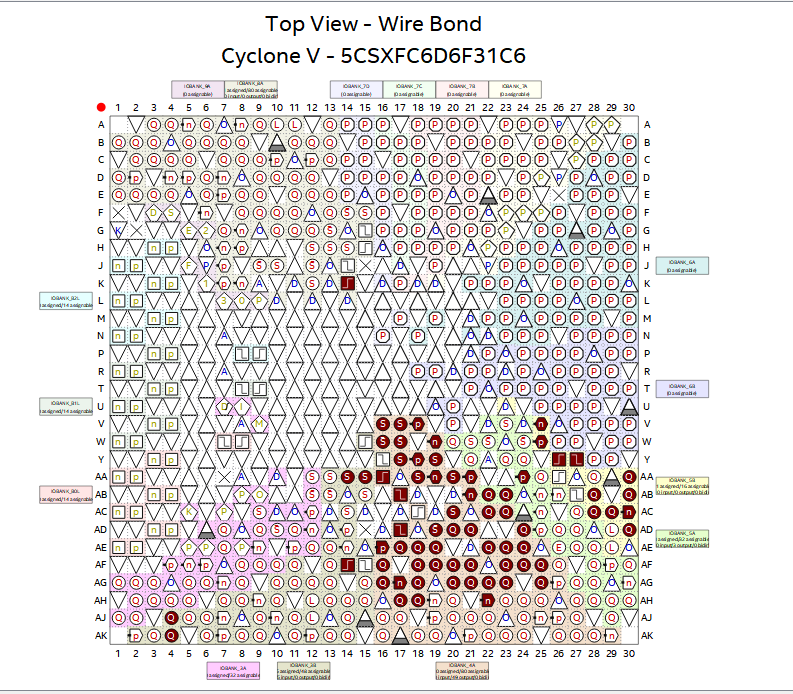
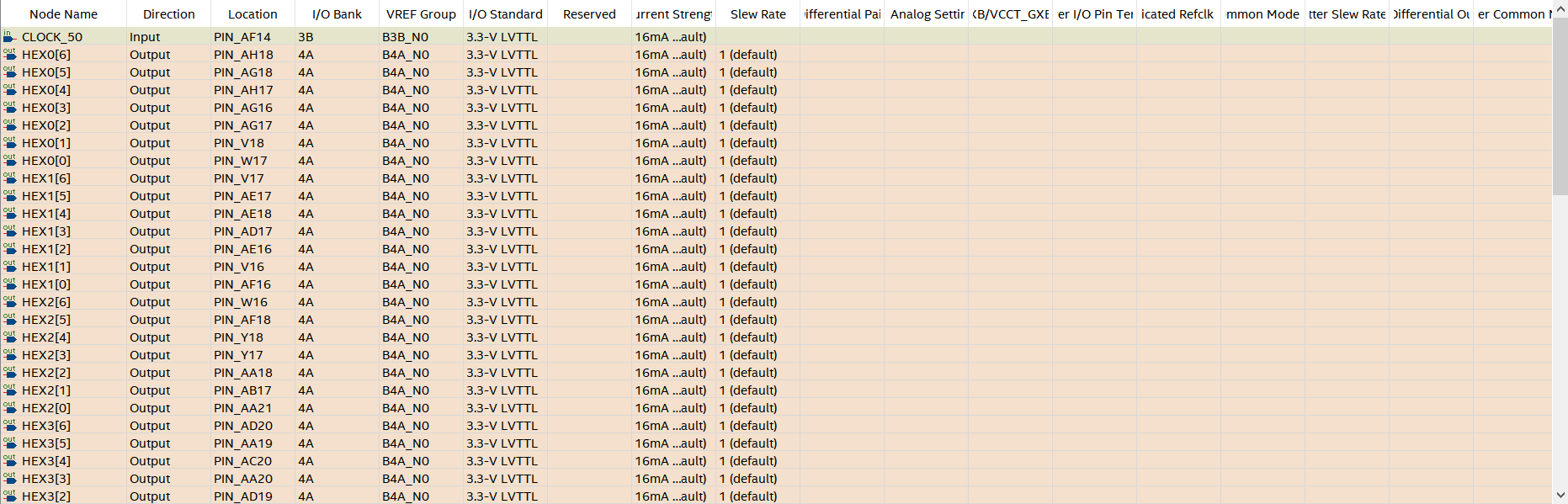


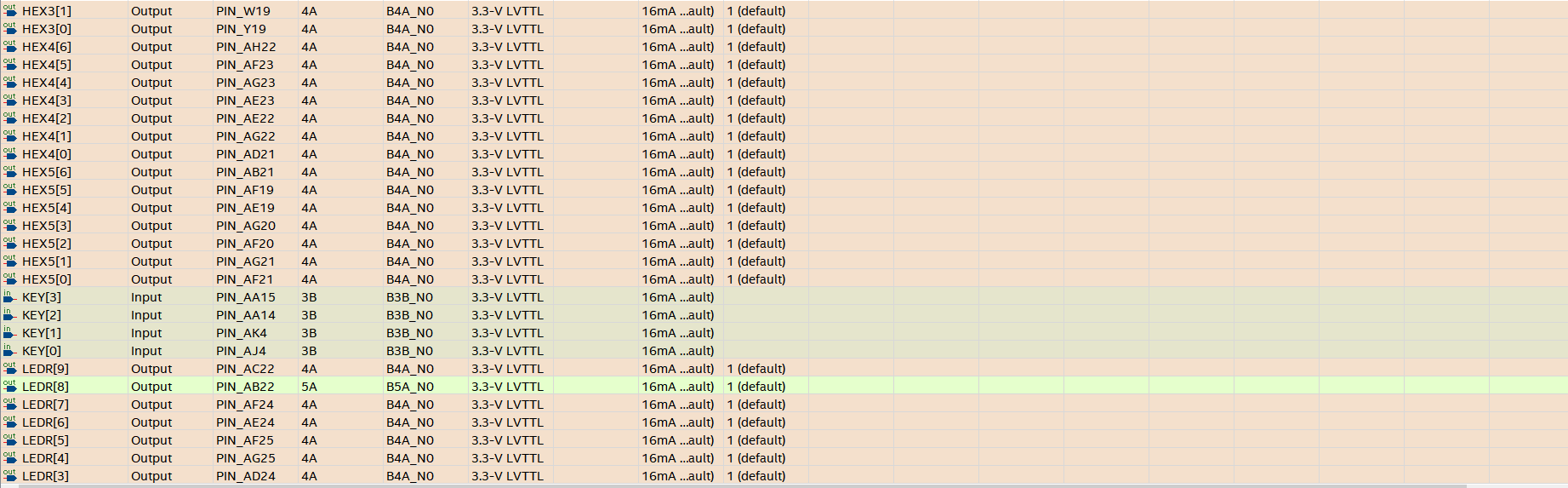
图 2-3-2:RTL视图

##### 2.4分配引脚

引脚分配使用DE10\_Standard\_SystemBuilder生成。







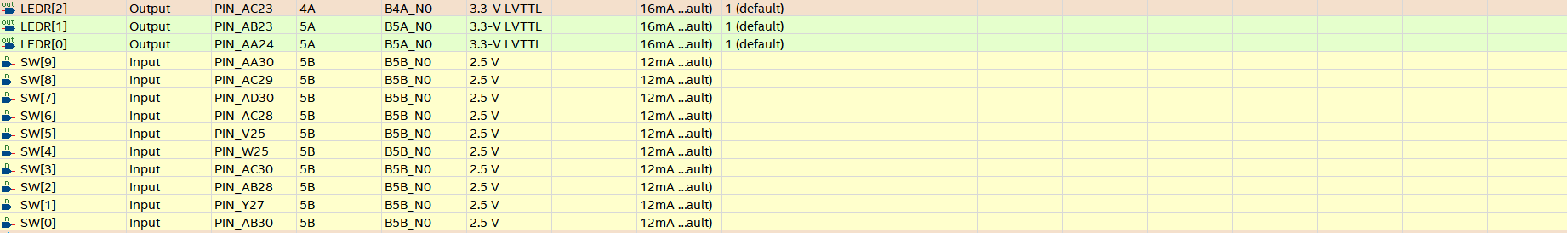


图2-5脚分配图

##### 2.5全编译

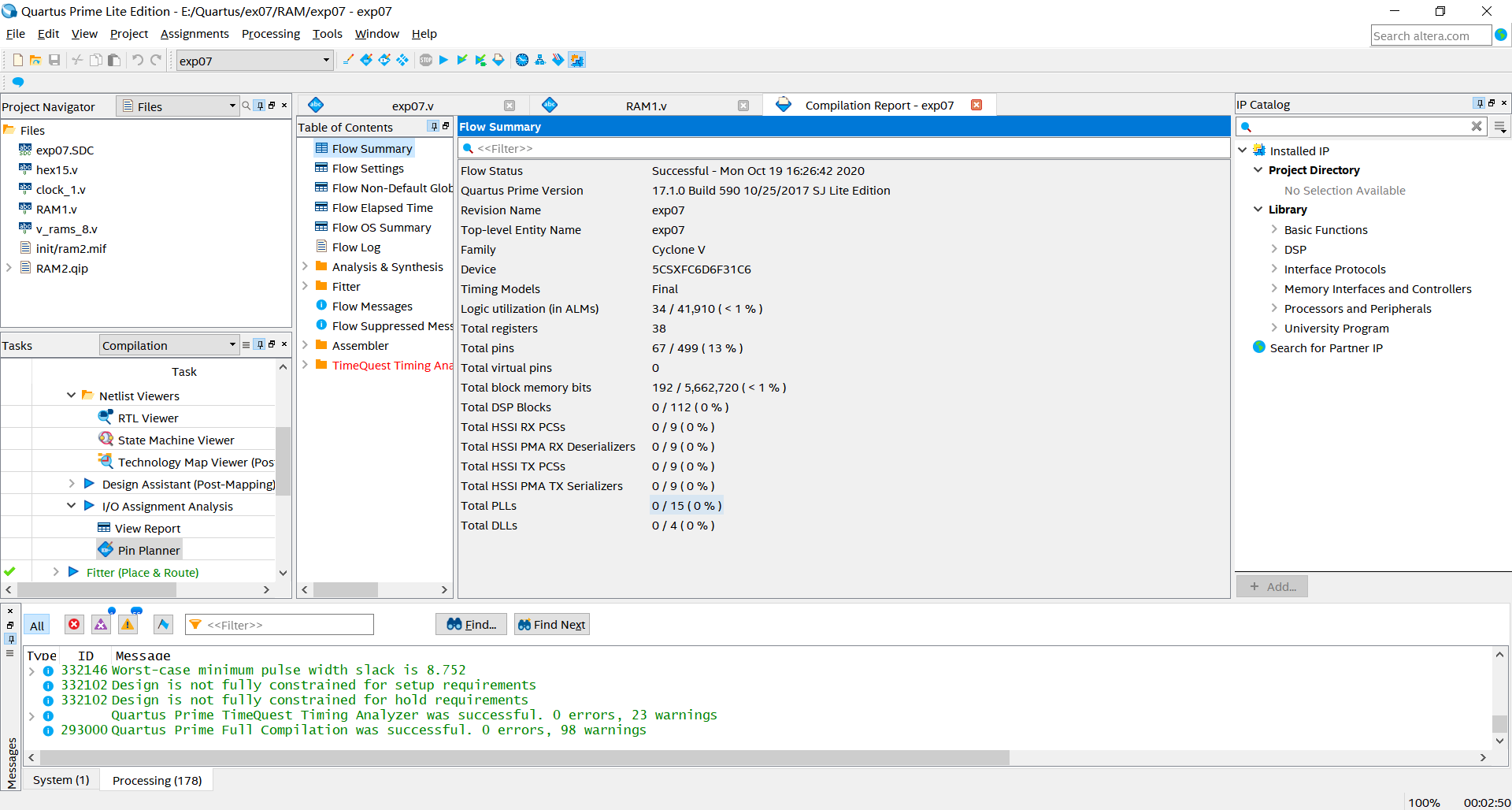


图2-6全编译成功

#### 实验总结

本次实验主要学习实践了存储器的设计、应用.还学习了不同的初始化方式——代码内初始化、外部txt文件初始化和.mif文件初始化. 通过对always语句和if语句的组合可以设计出不同读写模式的存储器.

本次实验还学习实践了使用IP核设计双口存储器,相比使用Verlilog语言来实现，该方法更加快捷、便利.

#### 附

##### 4.1 关于思考题

*如果将表 7 2中存储器实现部分改为*

1 always @(posedge clk)

2 if (we)

3 ram[inaddr] <= din;

4 else

5 dout <= ram[outaddr];

*该存储器的行为是否会发生变化？*

答：

该存储器的行为会发生改变.

修改部分对应exp07.pdf表7-2中的部分Verilog代码为：

15 always @(posedge clk)

16 if (we)

17 ram[inaddr] <= din;

18

19 assign dout = ram[outaddr];

实现的功能为，若写使能(we)有效(值为1),在时钟上升沿将din写入存储器的写地址(inaddr)对应内存单元中. dout的值在时钟的任何时刻都是读地址(outaddr)对应内存单元中的值.

修改后的代码实现的功能是：写使能有效时在时钟上升沿将din写入存储器的inaddr对应的地址,这一点和修改前一致, 但dout的值在写使能无效且时钟上升沿到来时才改变为outaddr对应存储器地址中的值,这是和原来代码实现功能的不同之处.

##### 4.2 关于存储器实例分析

exp07.pdf-7.2.1节中表7-3给出的存储器实例对应的RTL图如下：

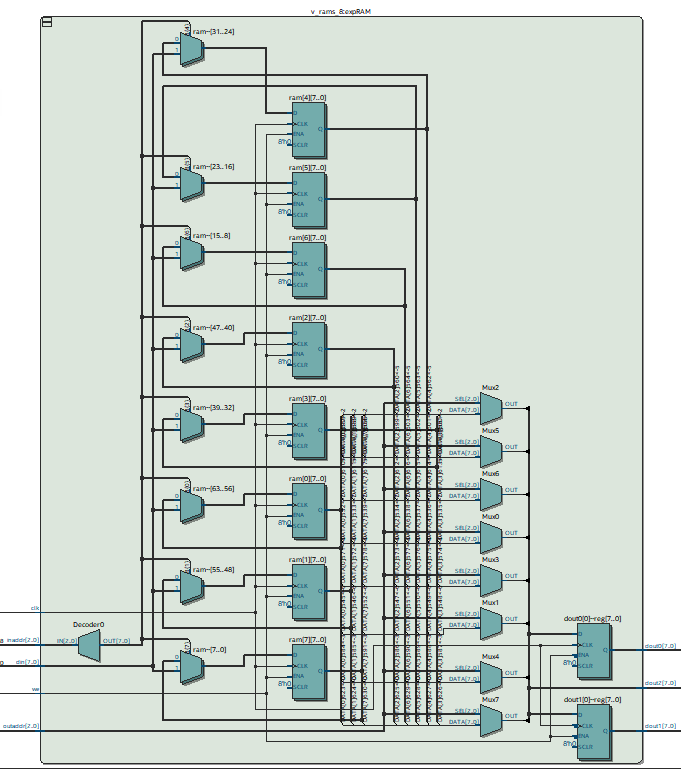


图4-2-1实例分析-RTL视图

通过分析xp07.pdf-表7-3中的dout0, dout1, dout2的赋值方式和RTL视图中对dout0, dout1, dout2输出的处理可以发现，dout0输出只有在写使能无效且时钟上升沿到来时才会改变为读地址输入(outaddr)对应内存地址中的数据;dout1输出只有在写使能无效且时钟下降沿到来时才会改变为读地址输入(outaddr)对应内存地址中的数据;dout2的赋值方式为阻塞式赋值，其变化与时钟变化无关,能直接得到读地址输入(outaddr)对应内存地址中的数据. 数据的输入则是在写使能信号有效且时钟上升沿到来时才会

该实例通过引脚约束后再开发板上验证得到的输入/输出时序与上面所述相同.

##### 4.3 1Hz分频器的Verilog实现

参照exp05.pdf表5-2秒时钟生成代码，本实验所用到的1Hz分频器的实现如下：

module clock\_1(clk\_50M, clk\_o);

input clk\_50M;

output reg clk\_o;

integer cnt;

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 | always @(posedge clk\_50M) begin  if(cnt == 25000000) begin  cnt <= 0;  clk\_o <= ~clk\_o;  end  else cnt <= cnt + 1;  end  endmodule |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |

##### 4.4 七段数码管编码器的Verilog实现

|  |  |
| --- | --- |
| 1  module hex15(x, y);  input [3:0]x;  output reg [6:0]y;    always @(\*) begin  case(x)  4'b0000:y = 7'b1000000;  4'b0001:y = 7'b1111001;  4'b0010:y = 7'b0100100;  4'b0011:y = 7'b0110000;  4'b0100:y = 7'b0011001;  4'b0101:y = 7'b0010010;  4'b0110:y = 7'b0000010;  4'b0111:y = 7'b1111000;  4'b1000:y = 7'b0000000;  4'b1001:y = 7'b0010000;  4'b1010:y = 7'b0001000;  4'b1011:y = 7'b0000011;  4'b1100:y = 7'b1000110;  4'b1101:y = 7'b0100001;  4'b1110:y = 7'b0000110;  4'b1111:y = 7'b0001110;  default: y = 7'b1111111;  endcase  end  endmodule |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |
| 23 |  |
| 24 |  |
| 25 |  |
| 26 |  |